

СПРАВОЧНИК ПО ФУНКЦИЯМ БИБЛИОТЕКИ DASH_library

ОПИСАНИЕ

DASH_library (версия 2.1) – библиотека из 350 функций для однопользовательского режима, выполненная в виде файлов **DASH_library.pbo** (основная часть) и **DASH_library_Settings.pbo** (необязательный файл стартовых настроек).

Все функции, настройки и значительная часть данных хранятся в **stringtable.csv**.
В тестовых миссиях функции условно разбиты на 20 тематик:

- [Первые шаги](#) – системные функции первой необходимости;
- [Геометрия](#) – задачи, где куда не плюнь – везде синусы;
- [Группы и юниты](#) – определение званий, позывных...;
- [Инвентарь](#) – оружие и боеприпасы;
- [Индексы](#) – махинации с индексами массивов;
- [Конвертация](#) – перевод одних величин в другие;
- [Маркеры](#) – операции с метками карты;
- [Массивы](#) – основные операции с массивами;
- [Математика](#) – разные математические функции и матричные операции;
- [Машинное обучение](#) – методы машинного распознавания образов;
- [Меню](#) – диалоговые окна;
- [Обработчики событий](#) – функции для совместной работы с Event Handlers;
- [Поиск](#) – поиск значений в массивах и базах данных;
- [Самодельные ОС](#) – скрипты, подражающие Event Handlers;
- [Системные](#) – наличие/тип переменных, вывод сообщений...;
- [Случайные числа](#) – генерация случайных чисел, выбор случайных элементов;
- [Сортировка](#) – изменение порядка элементов в массивах;
- [Среда обитания](#) – погода, свойства ландшафта и прочее;
- [Текст](#) – работа со строками и массивами строк;
- [Транспорт](#) – техника и её экипажи

НАЗНАЧЕНИЕ

Расширение возможностей скриптописателей и картоделов и экономия их времени на разработку непростых скриптов. Особенно для тех, кто не пишет SQF-функции сам.

Использование в моих собственных проектах, в том числе для командного движка.

ОТЛИЧИЯ ОТ ПРЕДЫДУЩЕЙ ВЕРСИИ (2.1 vs 2.0)

- ▶ 2 новые технологии хранения данных;
- ▶ 99% совместимость с ОФП 1.96;
- ▶ Добавлен файл стартовых настроек;
- ▶ 36 новых функций

УСТАНОВКА

Желательно наличие CWA 1.99, минимальная поддерживаемая версия ОФП – 1.96. В CWA часть кода выполняется быстрее, распознаются 100% островов и работает функция [overrideMoves](#).

Поместите файлы **DASH_library.pbo** и **DASH_library_Settings.pbo** в папку **Addons** в директории игры или в папку **Addons** какого-либо мода. Второй файл – необязательный, он нужен только если планируете менять стартовые опции.

Если нужно программно проверить наличие **DASH_library.pbo**, то есть простой способ:

```
"" != localize "dashlib"
```

Тестовые миссии забросьте в папку ...\\Users\\Ваш Профиль\\missions\\, а тестовую кампанию – в Campaigns

Если хотите использовать функции, требующие Fwatch 1.13+ – в поставке есть Fwatch 1.14, содержимое папки просто копируется в директорию установки игры. Предыдущая версия больше не поддерживается.

Сделайте ярлыки для необходимых модов, для чего в свойствах ярлыков прописывается что-то вроде: ...\\fwatchCWA.exe" -nosplash -nomap -mod=@ECP;@EKLMN;@EPRST, а для ОФП 1.96 ...\\fwatch.exe" -nosplash -nomap -mod=@ECP;@EKLMN;@EPRST

Кстати, официальный сайт автора: <http://ofp-faguss.com>

ЗАПУСК

В скрипт init.sqs или в строку инициализации игрока добавьте такой код:

```
call localize "dashlib"
```

Более подробная информация – в следующем пункте

ОПИСАНИЕ ФУНКЦИЙ

Первые шаги

call localize dashlib – запуск DASH_Library.

Должна запускаться ДО вызова любых других функций библиотеки, впрочем, некоторые из них могут работать и без запуска.

Создает и заполняет массив общих ресурсов **DASH**, массив ADD/CHANGE/REMOVE запросов **DASH_EN_QUEY** для работы с самодельными обработчиками событий, константу **sideUnknown**, а также несколько глобальных функций. Запускает скрипты, контролирующие погоду.

Удобно вызывать **dashlib** из init.sqs или в строке инициализации игрока.

Если строку **call localize "dashlib"** добавить в обработчик события «Init» в конфиге какого-то класса юнитов, то **dashlib** будет выполнена ровно 1 раз, сколько бы таких юнитов не было на карте.

При этом отсутствие DASH_Library.pbo не вызовет ошибку

При необходимости можно изменять стартовые опции библиотеки.

Для этого в **DASH_library_Settings.PBO** есть файл настроек **dashlib.ini**. Для частых экспериментов с настройками можно поменять к нему путь, указанный в **DASH_library_Settings\\stringtable.csv**, например, на корневую папку игры

```
Language, English
```

```
...
```

```
"dashlib_settings_path", "..\\dashlib.ini"
```

и поместить **dashlib.ini** в указанное место. Отсутствие необходимости запаковывать настройки сэкономит время.

Если файл **DASH_library_Settings.PBO** не изменен или вообще отсутствует, значения параметров будут такие:

```
_weather = true;  
_globalFns = ["alias", "loadfns", "setprojecttag", "print", "test"];  
_afterLaunch = {};
```

Расшифровка параметров.

При стандартном запуске опции погоды в редакторе будут сброшены, а «родные» команды **"setOvercast"** и **"setFog"** не будут работать. Если нужно отключить погодные скрипты - в файле **dashlib.ini** устанавливаем параметр **_weather=false**. Команды ОФП для облачности и тумана заработают в штатном режиме, но функции той же тематики (**fog**, **overcast**, **setOvercast2**, **setFog2**) работать не будут. Подробности в рубрике [Среда обитания](#)

Для изменения набора глобально видимых (загружаемых при старте) функций меняем параметр **_globalFns**, все имена должны быть в нижнем регистре

Также можно выполнить какой-нибудь код сразу же после запуска библиотеки, для этого меняем параметр **_afterLaunch**

function_name_in_lowercase call loadFns

альтернативный синтаксис:

ArrayOf_fns_names call loadFns - загружает одну или несколько функций из **stringtable.csv**, возвращает 2-х элементный массив:

0 - "OK", если все найдено, иначе строка-отчет об ошибках;

1 - массив имен загруженных функций.

Указанные в аргументах функции становятся глобально видимыми, названия должны указываться исключительно в нижнем регистре.

С помощью **loadFns** можно легко загрузить любые другие функции данного пака, НЕ добавляя в область глобальных переменных ничего лишнего, что защищает от конфликтов имен.

В тестовых примерах у меня это обычно делается «за кадром», чтобы не повторять по 250 раз одно и то же. Пример:

```
["debug", "hintdebug"] call loadFns
```

Будут созданы **debug** и **hintDebug**, полезные при отладке других функций и скриптов

correctVarName call setProjectTag - устанавливает глобальный тег, с которым будет работать **loadFns** - строка **correctVarName** будет добавлена перед названием каждой загружаемой ею функции. Возвращает true, если аргумент был строковым.

setProjectTag НЕ проверяет корректность тега, поэтому не стоит его начинать с цифры ("9k32_"). Тег, применяемый по-умолчанию - "". Назначение - предотвращает возможные коллизии с глобальными переменными других OFP проектов. Пример:

```
"FN_" call setProjectTag
```

Теперь все что загружается с помощью **loadFns**, будет начинаться с **"FN_"**

```
["splitbyrule", "inverse", "allvariants"] call loadFns
```

создаст функции **FN_splitByRule**, **FN_inverse** и **FN_allVariants**

Альтернативный подход – использование переименованных функций (псевдонимов):

[name_in_dashlib, name2] call alias - загружает функцию **name_in_dashlib** из **stringtable.csv** под новым именем **name2**. Позволяет использовать функции **DASH_library**, названия которых уже зарезервированы под другие переменные. Возвращает true, если функция найдена в таблице. Пример:

```
["pos2grid", "getMapQuad"] call alias
```

Есть еще один вариант применения функций DASH_Library - без «глобализации». В stringtable.csv все функции кроме **dashlib** хранятся под именами в формате "f_имявнижнемрегистре":

```
"Language", "English"
...
"f_iscwa", "!({getWorld} call localize {f_isnil})"
```

Применяем напрямую, без лишних переменных:

```
call localize "f_iscwa"
```

Именно в такой манере обычно вызываются функции внутри DASH_library

В тестовой миссии «Примеры (часть 1).Intro» в рубрике «Первые шаги» демонстрируются все вышеперечисленные методы
[Назад к списку тем](#)

Геометрия

[pos1, pos2, pos3] call **clockwiseOf** - вернет true если движение от pos1 до pos2 и потом от pos2 до pos3 происходит по часовой стрелке.
Другое применение: если первые 2 точки задают прямую, то результат true означает, что 3-я точка находится справа от прямой, false - слева

[pos1, pos2] call **delimPoint** - точка на плоскости между 2-я позициями, которая делит соединяющий позиции отрезок в заданном отношении (если пропорция не задана, то в качестве пропорции берется 1/1)

[Watcher, Target] call **dir2obj** - направление от юнита Watcher на объект Target

[Pos1, Pos2] call **dist2d** - расстояние между проекциями позиций на плоскость

[Pos1, Pos2] call **dist3d** - расстояние между позициями в трехмерном пространстве. Не совет, если позиции заданы относительно уровня моря (ASL)

[Pos1, Pos2] call **dirToPos** - направление в градусах от Pos1 до Pos2

Obj call **dirOfMove** - направление движения объекта Veh

Pos call **elevation** - высота точки над уровнем моря

[obj1, obj2] call **elevationAngle** - вертикальный угол от позиции объекта obj1 к объекту obj2. Если первый объект выше - результат (градусы) больше 0 и наоборот

[Watcher, Target] call **facingDiff** - разность в градусах между направлением корпуса юнита Watcher и направлением от него на объект Target

[Shelter, Guard, StepBack] call **hiddenPos** - найти позицию за укрытием Shelter на расстоянии StepBack от него, чтобы спрятаться от юнита Guard (предполагается, что Shelter достаточно велик и без проемов)

[pos, Range, Step] call **highestLowest** - самая высокая и самая низкая позиции над уровнем моря в квадрате с центром pos и стороной Range, шаг измерения Step

obj call **hrzSpeed** - горизонтальная составляющая скорости объекта(м/с)

[obj1, obj2] call **inFOV**

альтернативный синтаксис:

[obj1, obj2, FOV] call **inFOV** - true, если obj1 находится в поле зрения obj2 (не более чем 43 градуса по горизонтали в обе стороны от линии взгляда obj1)

– 3-м аргументом задаем свое поле зрения

[Pos1, Pos2] call **lineParams** – коэффициенты а и b уравнения прямой $Y=a*X + b$, проходящей через 2 позиции на плоскости

[Unit1, UnitArray] call **nearestFromArray**

альтернативный синтаксис:

[Pos1, UnitArray] call **nearestFromArray** – ближайший объект из UnitArray относительно юнита Unit1 / или позиции Pos1

[obj1, obj2] call **relPos2d** – относительная позиция obj2 в системе координат, связанной с объектом obj1 – начало на позиции obj1, ось Y совпадает с направлением корпуса obj1

[Obj, Height] call **setZ** – устанавливает объекту высоту над/под землей

[Pos1, Pos2, Pos3] call **triangleArea** – площадь треугольника, образованного на плоскости 3-я позициями. Если позиции расположены по-часовой стрелке, то площадь будет отрицательной (кому нужна только площадь – берите модуль (abs) от нее)

Практика: «Примеры (часть 2).Intro»

[Назад к списку тем](#)

Группы и юниты

unitArray call **any2Units**

альтернативный синтаксис:

groupTiger call **any2Units**

альтернативный синтаксис:

trigger1 call **any2Units** – переводит массив юнитов/группу/триггер в массив юнитов. Позволяет другим функциям принимать больше типов переменных

Unit call **boss** – юнит становится лидером в своей группе.

Как идея: игрок потерял половину бойцов – остальные взбунтовались и пошли за новым оленеводом)

[ClassName, pos, grp] call **createUnit2**

альтернативный синтаксис:

[ClassName, pos, grp, initString] call **createUnit2**

альтернативный синтаксис:

[ClassName, pos, grp, initString, skill] call **createUnit2**

альтернативный синтаксис:

[ClassName, pos, grp, initString, skill, strRank] call **createUnit2** –

очень похожа на команду "createUnit", но вдобавок возвращает ссылку на созданного бойца и позволяет в дальнейшем определять его ранг. Ничего не создаст и вернет objNull, если в отряде уже 12 юнитов

UnitArray call **enableQueryGroups** – создает внутренний массив групп для работы функции **getGroups**. Не нужна при использовании Fwatch 1.14+

UnitArray call **enableQueryRank** – позволяет определять ранг у всех юнитов в массиве UnitArray. Для юнитов, созданных с помощью **createUnit2** – не нужна

UnitArray call **filterGroups** – из списка юнитов (например, list someTrigger) создает массив представленных в нем групп

UnitArray call **getBoss** – возвращает персонажа из UnitArray с самым высоким званием. Если есть несколько офицеров одного ранга, будет выбран тот, у которого выше skill

Unit call **getGroupID**

альтернативный синтаксис:

Group call `getGroupID` - массив из названия (позывного) группы и цвета данного юнита/группы

Side call `getGroups`

альтернативный синтаксис:

SideArray call `getGroups` - список групп, соответствующих стороне или массиву сторон. Для работы без Fwatch требуется большой триггер типа Any с условием Present и функция **`enableQueryGroups`**. С Fwatch работает в редактируемых миссиях и в распакованных кампаниях

Group call `getGroupSpeed` - средняя скорость участников группы

Group call `getGroupSpread` - примерный диаметр группы

Unit call `getIntSide` - вернет 0 для east, 1 для west, 2 для resistance, 3 для civilian. Полезна для выбора из массива элементов, подходящих определенной стороне

Unit call `getRank` - воинское звание юнита в виде строки

Unit call `getRankID` - порядковый номер воинского звания юнита

[unitArray, surName] call `getUnitByLastName` - возвращает юнита с фамилией **surName**, если он есть в массиве, или **objNull** - в ином случае

Grp call `groupIsCargo` - проверяет, все ли юниты группы погрузились на транспорт

[Unit1, Unit2] call `isEnemy` - является ли **Unit2** видимым врагом **Unit1**

[UnitArray, Unit] call `joinSilent` - присоединить юнитов к отряду персонажа **Unit** без выдачи радиосообщений. Могут возникнуть проблемы с командой **`enableRadio`**, вместо неё применяйте **`enableRadio2`**

Grp call `mostInjured` - наиболее пострадавший из живых юнитов группы **Grp**

`sideUnknown` - добавленная сторона, к которой принадлежат снаряды, триггеры и **objNull**

unit call `squadNumber` - возвращает номер юнита в отряде - номер, по которому к нему обращается командир группы

GrpLeader call `squadOrder` - юниты **GrpLeader** в порядке возрастания их номеров. Лидер - всегда 0-й

[Unit1, Unit2] call `swapIdentity` - меняет местами личности (лица и голоса) юнитов **Unit1** и **Unit2**, повторное применение возвращает все на место. Для работы в пользовательских миссиях требуется saveGame в начале игры, в запакованных миссиях и кампаниях работает без проблем

[grp, pos] call `teleportGroup`

альтернативный синтаксис:

[grpLeader, pos] call `teleportGroup` - телепортировать всю группу на указанную позицию. При этом никто не покинет свой транспорт и будут сохранены расстояния между юнитами. Позиция лидера точно совпадет с точкой **pos**

[UnitArray, Angle] call `watchDir` - заставить всех юнитов смотреть в направлении **Angle** градусов.

Практика: «Примеры (часть 1).Intro», «Тестовая кампания»

[Назад к списку тем](#)

Инвентарь

Unit call ammo3 – модификация команды **ammo**. Возвращает 3-х компонентный вектор, показывающий число боеприпасов в магазине для каждого из 3-х слотов: первичное оружие, вторичное оружие и пистолет.

Если какой-то слот оружия пуст, то для него будет -1

[Unit, WeaponName] call cargoWeaponMagazines – все магазины к первому и второму стволам оружия **WeaponName**, находящиеся в грузовом отсеке техники или ящика **Unit**.
Работает исключительно в режиме кампании, чувствительна к регистру строк.
В случае неизвестного оружия – пустой массив

Man call emptySlots - количество пустых слотов под основное оружие (**primaryWeapon**, **secondaryWeapon**) у юнита **Man**

Man call emptySlots2 -возвращает 2 числа-количество пустых слотов под основное (**primaryWeapon**, **secondaryWeapon**) оружие и количество пустых пистолетных слотов у юнита **Man**

[UnitArray, Type] call getSpecialist

альтернативный синтаксис:

[Grp, Type] call getSpecialist – возвращает 1-го найденного юнита с экипировкой **Type**;
Type бывает 6-и видов: "RTO", "AA", "LD", "MG", "SNP", "AT"

[UnitArray,Type] call getSpecialists

альтернативный синтаксис:

[Grp,Type] call getSpecialists - возвращает всех найденных юнитов с экипировкой **Type**;
Type бывает 6-и видов: "RTO", "AA", "LD", "MG", "SNP", "AT"

Man call handgun - оружие из пистолетного слота юнита **Man**.

Юнит без пистолета --> пустая строка. Некоторые паки оружия могут добавлять вещи, не являющиеся пистолетом, но занимающие его слот, скажем, при использовании **SLX_Melee** понадобится дополнительная проверка на «пистолетность»

[UnitInfo, WeaponInfo, magazines] call invAdd

альтернативный синтаксис:

[UnitInfo, WeaponInfo, mag1, amount1, mag2, mag3, mag4, count4...] call invAdd – массово добавляет юнитам из **UnitInfo** заданное оружие и боеприпасы.

UnitInfo может быть юнитом, массивом юнитов, группой или триггером. **WeaponInfo** может быть названием оружия или массивом таких названий. Все элементы после этих двух рассматриваются как информация о магазинах

[UnitInfo, WeaponCargoInfo, MagazineCargoInfo] call invAddCargo - массово добавляет юнитам из **UnitInfo** заданное оружие и боеприпасы в грузовой отсек.

WeaponCargoInfo может выглядеть так:

["LAWLauncher", 3, "M16", 10, "AALauncher", "M60", 4],

если за названием не идет количество – используется значение 1, а может просто быть массивом имен классов оружия. **MagazineCargoInfo** имеет такую же структуру

[UnitInfo1, UnitInfo2] call invClone - копирует инвентарь из аргумента (0)

(юниты в форматах **UnitInfo**) и оснащает по их образцу юнитов аргумента (1).

Если число юнитов в **UnitInfo1** меньше, чем в **UnitInfo2**, то остальные юниты из **UnitInfo2** экипируются по образцу последнего юнита из **UnitInfo1**

Unit call invCreate

альтернативный синтаксис:

ClassName call invCreate – создает инвентарь - массив из массива оружия/экипировки данного юнита и массива его магазинов. Следует избегать массового вызова этой функции во втором варианте, так как он использует **selfcreate**. Полученный инвентарь можно использовать так : **[[UnitInfo]+Inventory] call invSet**

Unit call invCreate2

альтернативный синтаксис:

ClassName call **invCreate2** – функция, аналогичная **invCreate**, но возвращает в массиве оружия также «умения» вроде "Put" и "Throw"

UnitInfo call **invGet** – создает инвентарь – массив из массива оружия/экипировки данных юнитов/юнита/группы/всех найденных триггером, заданных **UnitInfo**, и массива магазинов

UnitInfo call **invGet2** – аналогична **invGet2**, но возвращает в массиве оружия также «умения» вроде "Put" и "Throw"

[**UnitInfo**, **WeaponInfo**, **magazines**] call **invOwners**

альтернативный синтаксис:

[**UnitInfo**, **WeaponInfo**, **mag1**, **amount1**, **mag2**, **mag3**, **mag4**, **count4...**] call **invOwners** – ищет среди юнитов **UnitInfo** обладателей данного оружия и боеприпасов.

UnitInfo может быть юнитом, массивом юнитов, группой или триггером. **WeaponInfo** может быть названием оружия или массивом таких названий. Все элементы после этих двух рассматриваются как информация о магазинах

[**UnitInfo**, **WeaponInfo**, **magazines**] call **invRemove**

альтернативный синтаксис:

[**UnitInfo**, **WeaponInfo**, **mag1**, **amount1**, **mag2**, **mag3**, **mag4**, **count4...**] call **invRemove** – массово удаляет у юнитов из **UnitInfo** заданное оружие и боеприпасы.

UnitInfo может быть юнитом, массивом юнитов, группой или триггером. **WeaponInfo** может быть названием оружия или массивом таких названий. Все элементы после этих двух рассматриваются как информация о магазинах

[**Unit**, **WeaponList**, **MagazineList**] call **invQueryCargo** – возвращает массив количества единиц каждого оружия из **WeaponList** и массив количества единиц каждого магазина из **MagazineList** в грузовом отсеке юнита. Работает исключительно в режиме кампании

[**UnitInfo**, **WeaponInfo**, **magazines**] call **invSet**

альтернативный синтаксис:

[**UnitInfo**, **WeaponInfo**, **mag1**, **C1**, **mag2**, **mag3**, **mag4**, **C4**, **mag5...**] call **invSet** – массово устанавливает юнитам из **UnitInfo** заданное оружие и боеприпасы.

UnitInfo может быть юнитом, массивом юнитов, группой или триггером. **WeaponInfo** может быть названием оружия или массивом таких названий. Все элементы после этих двух рассматриваются как информация о магазинах

[**UnitInfo**, **WeaponCargoInfo**, **MagazineCargoInfo**] call **invSetCargo** – массово устанавливает юнитам из **UnitInfo** заданное оружие и боеприпасы в грузовой отсек.

WeaponCargoInfo может выглядеть так:

["LAWLauncher", 3, "M16", 10, "AALauncher", "M60", 4],

если за названием класса не идет количество – используется значение 1, а может просто быть массивом имен классов оружия. **MagazineCargoInfo** имеет такую же структуру

Unit call **isRTO** – true, если у **Unit** есть радио

Unit call **isMGSoldier** – true, если у **Unit** есть пулемет

Unit call **isSniper** – true, если у **Unit** есть снайперская винтовка

Unit call **isLaserSoldier** – true, если у **Unit** есть лазерный дальномер

Unit call **isAASoldier** – true, если у **Unit** есть ПЗРК

Unit call **isATSoldier** – true, если у **Unit** есть тяжелый гранатомет/ПТРК

Все функции «mag*», кроме **magSize**, чувствительны к регистру строк!

MagazineName call **magAmmoAir** – название класса пуль в магазине, подходящего для атаки воздушных целей (в конфиге в классе **cfgAmmo** параметр **airLock=1**).

Пустая строка, если нет данных или нет такой пули/снаряда

MagazineName call **magAmmoInf** – название класса пуль в магазине, подходящего для атаки живой силы (в конфиге в классе **cfgAmmo** параметр **irLock=0**).

Пустая строка, если нет данных или нет такой пули/снаряда

MagazineName call **magConvert** – преобразует магазины с повышенным разбросом (HD) из **JAM_Magazines.pbo** или мода **@WW4 2.5** в соответствующие магазины с обычным разбросом и наоборот. Пустая строка, если нет данных или если у магазина нет HD-версии

MagazineName call **magLightColor** – определяет цвет сигнальной ракеты. Пустая строка, если нет данных или если снаряды магазина не являются сигнальными ракетами

MagazineName call **magRangeInfo** – возвращает массив:

0 – минимальный радиус действия пуль/снарядов из **MagazineName**;

1 – максимальный радиус действия пуль/снарядов из **MagazineName**;

2 – класс пуль для минимума;

3 – класс пуль для максимума

Если магазин неизвестен – вернет пустой массив

MagazineName call **magSize** – определяет, сколько слотов занимает магазин данного типа. Не нуждается в базе данных, дополнение с этим магазином должно быть загружено

MagazineName call **magSlotInfo** – возвращает массив:

0 – тип магазина (0, 32 или 256 – не занимающие слотов, пистолетный и обычный).

1 – количество занимаемых слотов

Использует базу данных, может проверять магазины даже не установленных аддонов

MagazineName call **magSmokeColor** – определяет цвет дымовой шашки. Пустая строка, если нет данных или если снаряды магазина не являются дымовыми

Man call **radioState** – проверка состояния рации, >0: рация работает, 0: нет заряда батарей, -1: нет рации

[Unit, WeaponName] call **unitWeaponMags** – массив из двух элементов:

0 – все классы магазинов к первому стволу **WeaponName**, которые есть у **Unit**;

1 – все классы магазинов ко второму стволу **WeaponName**, которые есть у **Unit**;

Обычно второй массив – пустой, для автоматов с подствольником в нем будут гранаты и сигнальные ракеты

WeaponName call **weaponIsSilent** – true, если оружие бесшумное

WeaponName call **weaponMags** – массив из двух элементов:

0 – все классы магазинов к первому стволу **WeaponName**;

1 – все классы магазинов ко второму стволу **WeaponName**, обычно пустой массив.

Пустой массив, если оружие не найдено в базе

Практика: «Примеры (часть 3).Intro», «Тестовая кампания»

[Назад к списку тем](#)

Индексы

[Array, Index] call **idxDelete** – удалить из массива элемент с данным индексом.

Не возвращает результат – меняет исходный массив

[Array, IntegerArray] call **idxDeleteM** – удалить из массива элементы с индексами из **IntegerArray**. Не возвращает результат – меняет исходный массив

NumericArray call **idxMax** – найти индекс максимального элемента в массиве

NumericArray call **idxMin** - найти индекс наименьшего элемента в массиве

[Array, Index] call **idxToEnd**

альтернативный синтаксис:

[Array, ArrayOfIndexes] call **idxToEnd** - элемент(ы) с указанными номерами переместить в конец массива **Array**. Не возвращает результат – меняет исходный массив

[Array, Index] call **idxToStart**

альтернативный синтаксис:

[Array, ArrayOfIndexes] call **idxToStart** - элемент(ы) с указанными номерами переместить в начало массива **Array**. Не возвращает результат – меняет исходный массив

[Array, IntegerArray] call **idxTake** - выбрать элементы массива **Array**, находящиеся на позициях, заданных массивом целых чисел **IntegerArray**. Возвращает массив

Практика: «Примеры (часть 1).Intro»

[Назад к списку тем](#)

Конвертация

Angle call **angle2compass** - направление в градусах переводит в направление по сторонам света

Angle call **angle2hour** - направление в градусах переводит в часовой угол (1-12)

Array call **any2pos**

альтернативный синтаксис:

SomeObject call **any2pos**

альтернативный синтаксис:

SomeGroup call **any2pos**

альтернативный синтаксис:

MarkerName call **any2pos** – возвращает 3-D позицию. Применяется в других функциях, позволяя им работать с аргументами разного типа

[FloatValue, StrOld, StrNew] call **convert** - перевести **FloatValue** из одной меры длины в другую. Доступные меры длины: "ml", "yd", "ft", "in", "cm", "m", "km" – мили, ярды, футы, дюймы, сантиметры, метры, километры

StrQuad call **grid2pos** - название квадрата (строка) - в координаты карты [X,Y].

От регистра строки **StrQuad** не зависит.

При использовании модов @WGL или @Invasion44, координатная сетка большинства островов становится 6-цифровой, функция **grid2pos** автоматически переходит на нужный формат

Str call **mtime2daytime** – строка с натовским обозначением времени типа "0730" - в реальное время из диапазона(0.0000-23.9999)

pos call **pos2grid** – позицию **pos** - в название квадрата карты (строку вроде "E58").

При использовании модов @WGL или @Invasion44, на большинстве островов возвращает строку из 6 цифр ("114098")

[dTime, TimeFormat] call **time2string** - форматировать время **dTime**

(в часах) согласно формату **TimeFormat**. Доступны форматы: "HH", "HHMM", "HH:MM", "HH:MM:SS", "HH:MM;SS:MM".

Если **TimeFormat** не задан - используется "HHMM" («военное время» в амерской армии)

Практика: «Примеры (часть 2).Intro»

[Назад к списку тем](#)

Маркеры

[MarkerName, Delay] call **animateMarker**

альтернативный синтаксис:

[MarkerName, Delay, ScaleFactor] call **animateMarker** – заставляет MarkerName пульсировать. ScaleFactor – во сколько раз маркер будет изменяться в размерах. Может принимать значения от 0 до 1000. Если меньше 1 – маркер будет уменьшаться. Если не задан явно, то считается равным 0.5

[Unit, MarkerName] call **attachMarker**

альтернативный синтаксис:

[Unit, MarkerName, Delay] call **attachMarker** – прикрепить маркер к юниту. Опционально можно задать, через сколько секунд проверяется позиция юнита

MarkerName call **detachMarker** – останавливает скрипт, привязывающий MarkerName к какому-либо юниту. После применения необходимо подождать пару секунд перед новой привязкой маркера к юниту

MarkerType call **getMissionMarkers**

альтернативный синтаксис:

MarkerTypeArray call **getMissionMarkers** – работает только с Fwatch 1.13+, возвращает имена всех маркеров на карте (кроме пользовательских), имеющих заданный тип

MarkerName call **hideMarker** – прячет маркер, уменьшая его размер в 5000 раз

[MarkerName1, Pos] call **markerDist**

альтернативный синтаксис:

[MarkerName1, Unit] call **markerDist**

альтернативный синтаксис:

[MarkerName1, MarkerName2] call **markerDist** – расстояние от метки на карте MarkerName1 до позиции или до юнита или до другого маркера

[Pos, MarkerArray] call **nearestMarker**

альтернативный синтаксис:

[Pos, MarkerArray, MaxDistance] call **nearestMarker** – найти ближайший к Pos маркер из массива MarkerArray. Опционально задается максимальный радиус поиска

MarkerName call **showMarker** – увеличивает скрытый маркер в 5000 раз, делая его снова видимым

MarkerName call **stopMarkerAnim** – останавливает скрипт, анимирующий MarkerName. После применения необходимо подождать пару секунд (~2) перед заданием маркеру новой анимации

call **userMarkers** – возвращает массив имен всех пометок, сделанных игроком на карте в ходе миссии. Метки карты, удаленные игроком, не входят в этот массив

Практика: «Примеры (часть 1).Intro»

[Назад к списку тем](#)

Массивы

Array call **allSubsets** – все подмножества массива, не включая массив полностью. Ограничение на длину аргумента – не более 13

ArrayOfArrays call **allVariants** – все комбинации: из

[[true,false], [1,2,3], ["a", "b"]] call allVariants

получится массив из 12 подмассивов от [true, 1, "a"] до [false, 3, "b"]

Array call **array2Pairs** – массив вида [name1, v1, name2, v2, name3, v3] преобразуется к виду [[name1, v1], [name2, v2], [name3, v3]].

Хорошо сочетается с **addWeaponCargo** / **addMagazineCargo**.

Позволяет создавать функции с произвольным порядком аргументов:

```
["radius", 2000, "pos", getPos boom, "EMP", true] call nuclearExplosion ==  
["EMP", true, "pos", getPos boom, "radius", 2000] call nuclearExplosion
```

[Arr1, Arr2] call **arrayAddArray** – сложить массивы поэлементно. Результат – массив

[Arr, Nmb] call **arrayAddNumber** – прибавить число к каждому элементу массива.
Результат – массив

[Arr1, Arr2, expr] call **arrayFncArray** – применить **expr** к каждой паре соответствующих элементов двух массивов. Результат – массив

[Arr, Var, expr] call **arrayFncScalar** – применить выражение **expr** к каждой паре [элемент массива Arr, Var]. Результат – массив

NestedArray call **arrayMean** – средние значения вложенного массива **NestedArray** по каждому измерению. Возвращает массив той же длины что и (**NestedArray** select 0)

[Arr1, Arr2] call **arrayMultArray** – поэлементное перемножение массивов.
Результат – массив

[Arr1, Arr2] call **arraysAreIntersecting**

альтернативный синтаксис:

[Arr1, Arr2, ignoreCase] call **arraysAreIntersecting** – проверка массивов на наличие хотя бы 1 общего элемента.

Если задать **ignoreCase=true**, то функция не будет обращать внимание на регистр сравниваемых строк. Это будет иметь смысл, если проверяется наличие у юнита любого из указанных стволов/магазинов и нет уверенности, что в заданном массиве все они написаны в правильном регистре

ArrayOfArrays call **arraySimplify** – уменьшить глубину вложенности массива на 1. Массив массивов станет простым массивом, при этом наличие в нем скаляров не вызовет ошибки. В случае простого массива вернет его копию

[Arr1, Arr2] call **arraysIntersection**

альтернативный синтаксис:

[Arr1, Arr2, ignoreCase] call **arraysIntersection** – получить элементы массива **Arr1**, входящие в массив **Arr2**.

Если задать **ignoreCase=true**, то функция не будет обращать внимание на регистр сравниваемых строк, что полезно при работе с названиями классов – оружия, магазинов, пуль и юнитов

[Array1, Array2] call **compareArrays** – сравнивает 2 массива поэлементно, выдает true, если они равны. Функция нечувствительна к регистру строк, но массивы не могут содержать элементы разного типа или булевы величины/подмассивы.

Массивы разной длины – всегда не равны

[Array1, Array2] call **compareHeaps** – сравнивает 2 массива, не учитывая порядок элементов. Чувствительна к регистру строк, но позволяет сравнивать массивы из разнородных и даже булевых элементов. Например, позволяет определить изменился ли с прошлого запроса набор магазинов в инвентаре бойца – порядок магазинов тут безразличен

Var call **isArray** – возвращает true, если **Var** – массив. Еще можно применять проверку на не-массив: **_this in [_this]**

[start, end, step] call **makeArray**

альтернативный синтаксис:

[start, end] call **makeArray** – массив равномерно возрастающих/убывающих чисел, начиная со **start**, не превышающих **end**, с шагом **step**. По-умолчанию шаг прогрессии=1

Array call **pop** – удалить из **Array** последний элемент и вернуть его

[Integer, Any] call **populate** - создать массив из Integer штук элементов Any.
Если Any - массив, то делаются копии его содержимого, а не копируются ссылки на него

NumericArray call **product** - произведение элементов массива

[Array, element] call **pushForward** - добавить элемент в начало массива.
Не возвращает результат – меняет исходный массив

[Array, element] call **pushNew** - если элемента нет в массиве, то он будет добавлен в его хвост. В случае успеха возвращает true

[Array, Old, New] call **replace** - заменяет в массиве все вхождения элемента Old на элемент New. Возвращает новый массив

[Array, ArrOld, ArrNew] call **replaceM** - в массиве Array заменяет каждое вхождение элемента из массива ArrOld соответствующим ему элементом из ArrNew.
Возвращает новый массив

[Array, Condition] call **splitByRule** - разделить Array на 2 части по условию Condition.
В первый подмассив попадут элементы, для которых Condition выполняется, во второй - остальные)

Array call **split1212** - разделить Array на 2 части.
В первый подмассив попадут нечетные элементы, во второй – четные.
Если массив был вида [key1, val1, key2, val2, key3, val3...], то после применения ключи и значения будут в разных подмассивах

[Array, ArrayOfArrays] call **splitByArrays** - «раздать» элементы Array массивам из ArrayOfArrays по круговой системе. Поможет разделить некоторый общий ресурс между группами или разбить отряд на несколько отделений

NumericArray call **sum** - сумма элементов массива;

[Array, N] call **take** - взять N первых элементов массива Array, если N>0, или взять N последних, если N<0;

ArrayOfArrays call **transpose** - меняет местами в матрице строки со столбцами.
Пример:

```
[[1,2,3], [4,5,6]] call transpose  
результат - [[1,4], [2,5], [3,6]]
```

Array call **typeList** - массив уникальных элементов и соответствующий массив количества элементов каждого типа в массиве Array. Подойдет для рапортов и скриптов учета ресурсов

Array call **unique** - получить уникальные (неповторяющиеся)элементы из массива.
Массив может содержать числа, строки, объекты и группы, быть однородным или из элементов разного типа (кроме массивов). Функция чувствительна к регистру строк

[Vals, Numbers] call **unwrapList** – строит простой массив, содержащий
Numbers[0] элементов Vals[0], Numbers[1] элементов Vals[1]...
Выполняет задачу, обратную работе функции **typeList**

[NumArray1, NumArray2] call **without**
[StringArray1, StringArray2] call **without** - альтернатива стандартному вычитанию массивов. Пример:
[["m4", "m4", "m60"], ["m4", "m21"]] call without
вернет ["m4", "m60"]

Практика: «Примеры (часть 1).Intro»
[Назад к списку тем](#)

Математика

Number call **ceil** – минимальное целое число, не меньшее **Number** (округление вверх)

[Matrix, I] call **column** – выбрать из матрицы I-й столбец

[Matrix, IntArray] call **columns** – выбрать столбцы матрицы, перечисленные в **IntArray**.
Вернет матрицу с меньшей длиной подмассива

Int call **dec2bin** – натуральное число – в массив нулей и единиц

QuadMatrix call **determinant** – определитель (квадратной) матрицы – вложенного массива с одинаковой длиной всех строк, равной их числу. Применяется во многих задачах алгебры и вычислительной геометрии

Int call **digits** – массив цифр, из которых состоит число

Number call **floor** – максимальное целое число, не превышающее **Number** (округление вниз)

[Number, [min1, max1], [min2, max2]...] call **isInRanges**

[Number, [min1, max1, min2, max2...] call **isInRanges** – вернет true, если **Number** попадает в один из диапазонов

[Number, Base] call **log2arg** – логарифм числа **Number** по основанию **Base**

NumberArray call **mean** – среднее арифметическое чисел массива

[Matrix, I, J] call **minor** – получить новую матрицу путем удаления строки **I** и столбца **J** из старой. Если **I** или **J** меньше 0, то соответствующая строка или столбец останутся на месте

Array call **mode** – самый частый («модный») элемент в массиве

[Any, Nrows, Ncols] call **mtxBuild**

альтернативный синтаксис:

[Array, Nrows, Ncols] call **mtxBuild**

альтернативный синтаксис:

Array call **mtxBuild** – создать матрицу – массив из подмассивов равной длины.

В первом случае создается матрица из **Nrows** строк (подмассивов) и **Ncols** столбцов (это длина подмассива), заполненная элементом **Any**.

Во втором случае аналогичная матрица будет заполнена значениями из **Array**.

В третьем варианте, если длина массива == **N*N** (квадрат целого числа) то будет построена матрица в **N** строк и **N** столбцов (квадратная)

Matrix call **mtxCopy** – создать независимую копию данной матрицы

QuadMatrix call **mtxInverse** – обращение (квадратной) матрицы – вложенного массива с одинаковой длиной всех строк, равной их числу. Пригодится в задачах вычислительной геометрии

[Matrix1, Matrix2] call **mtxProduct** – произведение двух матриц по правилам линейной алгебры

Number call **round**

альтернативный синтаксис:

[Number, NumOfDigits] call **round** – округлить число до целых либо с указанной точностью

Number call **sign** – знак числа: -1 для отрицательных, 0 для 0, 1 для положительных чисел

[Array1, Array2] call **vectorProduct** – сумма попарных произведений двух массивов

[bool1, bool2] call **xor** – «исключающее или», аналог $A \neq B$ для булевых величин

Практика: «Примеры (часть 3).Intro»

[Назад к списку тем](#)

Машинное обучение

[Data, ClassLabels] call **ANN1Learn**

альтернативный синтаксис:

[Data, ClassLabels, Ages] call **ANN1Learn** – простая однослойная нейронная сеть, решающая с приличной скоростью несложные задачи классификации.

Data – массив массивов (точек в пространстве признаков).

Длина каждого подмассива==число признаков.

Массив **ClassLabels** содержит единицы и двойки – метки первого и второго классов распознаваемых объектов.

Ages – число циклов обучения, по-умолчанию = 200.

Результирующий массив: веса каждого признака + [вес смещения]

[Weights, NewPoint] call **ANN1Use** – использовать посчитанный массив весов **Weights**, чтобы решить, к какому классу отнести точку **NewPoint**.

Возвращает 1 или 2

[Weights, NewPoints] call **ANN1UseM** – использует массив весов для классификации массива неопознанных точек **NewPoints**.

Возвращает массив единиц и двоек

Array_0_1 call **boolNot** – каждая единица в массиве становится нулём и наоборот

[Array_0_1, anotherArray_0_1, strCode] call **boolOperation** – провести булеву операцию между каждой парой соответствующих элементов массивов.

Массивы должны состоять из нулей и единиц. **StrCode** может принимать 8 разных значений.

У каждой операции из списка есть синоним:

"and" - "&&", "or" - "||", "xor" - "!= cant", "equal" - "==" cant

Следующие 6 функций представляют собой меры различия двух точек во многомерном пространстве некоторых численных признаков.

Все кроме **h_dist** работают только с числовыми данными.

Все кроме **cos_dist** возвращают значения в интервале [0,+inf]

Результат – число, причем при несовпадении длин векторов это «+бесконечность»

[NumArray1, NumArray2] call **che_dist** – «шахматное» расстояние или расстояние Чебышева. Равно максимальному из попарных расстояний между элементами векторов.

Не отличается большой точностью, зато вычисляется быстро

[NumArray1, NumArray2] call **cos_dist** – косинус-расстояние между массивами.

Употребляется в нечетком текстовом поиске. Если **NumArray1** и **NumArray2** – прямые, заданные двумя точками каждая, то $1 - ([NumArray1, NumArray2] call cos_dist)$ вернет косинус острого угла между ними

[NumArray1, NumArray2] call **e_dist** – евклидово расстояние между массивами. Одно из самых популярных в задачах машинного обучения

[Array1, Array2] call **h_dist** – расстояние Хемминга между массивами.

Фактически – число несовпадений элементов. Самое быстрое из всех расстояний.

Массивы могут содержать любые значения одного типа кроме булевых и массивов, но все же рекомендуется использовать целые числа либо строки

[NumArray1, NumArray2] call **m_dist** – «манхеттенское» или «квартальное» расстояние между массивами. Одно из быстрых для вычисления

[NumArray1, NumArray2] call **k_dist** – К-мера. Взвешенный вариант **m_dist**.

Применять, если предыдущие функции почему-то не подошли

[Data, Labels, K] call **etalonLearn**

альтернативный синтаксис:

[Data, Labels, K, DistFunction] call **etalonLearn** – создает массив эталонов длины K, полученных сжатием Data методом «К средних» и массив меток их классов.

Две следующие функции могут использовать полученный результат для определения меток классов (классификации) неизвестных точек

[Etalons, ClassLabels, Point] call **etalonUse**

альтернативный синтаксис:

[Etalons, ClassLabels, Point, DistFunction] call **etalonUse** – определить метку класса точки Point по метке класса ближайшего к ней «эталона» – вариация метода «ближайшего соседа». DistFunction определяет, каким выражением определяется расстояние между точками, по-умолчанию используется **e_dist**

[Etalons, ClassLabels, Points] call **etalonUseM**

альтернативный синтаксис:

[Etalons, ClassLabels, Points, DistFunction] call **etalonUseM** – определить метки классов точек Points по меткам классов ближайших к ним «эталонов» – вариация метода «ближайшего соседа». DistFunction определяет, каким выражением определяется расстояние между точками, по-умолчанию используется **e_dist**

[Array, Condition] call **getMatchArray** – возвращает массив с 1 на месте тех элементов Array, для которых условие Condition верно, и 0 – на месте остальных

[Data, K] call **kmeans**

альтернативный синтаксис:

[Data, K, DistFunction] call **kmeans** – определить методом «К средних» центры K кластеров (скоплений), образованных многомерными данными Data. Помогает уменьшить число данных для обучения других алгоритмов. При слишком малой длине набора Data или слишком большом K наблюдаются ошибки кластеризации – будет исправлено в следующей версии

DistFunction определяет, каким выражением определяется расстояние между точками. По-умолчанию используется **e_dist**

[Array, Xpression] call **makeDataRecords** – вызвать Xpression для каждого элемента в Array, результат выдать в виде массива.

Результат напоминает записи базы данных.

Пример Xpression: **{_x knowsabout player}, {getDammage _x, skill _x}**

Внимание: не злоупотребляйте в Xpression квадратными скобками – выражение заключается в скобки автоматически.

Создана, чтобы собирать с массива юнитов набор данных для обучения или распознавания

[Array1, Array2] call **match** – процент совпадения между соответствующими элементами двух массивов. Назначение: при контролируемом обучении – сравнить правильные метки классов с результатом классификации тех же данных алгоритмом машинного обучения

NumArray call **scale01** – масштабировать массив, чтобы элементы лежали в диапазоне [0...1]

[NumArray, A, B] call **scaleMinMax** – масштабировать массив, чтобы элементы лежали в диапазоне [A...B]

[Data, Labels] call **split4LearnAndTest**

альтернативный синтаксис:

[Data, Labels, ratio] call **split4LearnAndTest** – случайным образом разделить данные на обучающую и тестовую выборки. Можно задать процент или долю точек для обучения, рекомендуемый диапазон – [25...75] или [0.25...0.75], по-умолчанию данные делятся примерно пополам. Результат – массив из 4 элементов:

0 – данные для обучения (матрица)

1 – метки классов для обучения (массив)

- 2 - данные для тестирования (матрица)
- 3 - метки классов для тестирования (массив)

Практика: «Примеры (часть 1).Intro»

[Назад к списку тем](#)

Меню

[minIndex, maxIndex] call **allCtrls** – найти idc всех видимых элементов меню. Поиск ведется от minIndex до maxIndex включительно. Возвращает массив

[idc, StringArray] call **fillBox** – очищает и затем заполняет ListBox или ComboBox idc строками из массива StringArray

Практика: «Примеры (часть 1).Intro»

[Назад к списку тем](#)

Обработчики событий

[UnitInfo, oldAnim1, newAnim1, oldAnim2, newAnim2...] call **overrideMoves** - массово переопределяет анимации юнитов, в том числе игрока.

UnitInfo – юнит, массив юнитов, группа или триггер.

Хорошо работает с перемещениями, с остальными playMove анимациями не очень-то дружит, нередко выполняясь не вместо а ПОСЛЕ них. Требует CWA

[UnitInfo, oldAmmo1, newAmmo1, oldAmmo2, newAmmo2...] call **replaceBullets** - массово заменяет выстреливаемые пули/снаряды/ракеты снарядами других типов.

UnitInfo – юнит, массив юнитов, группа или триггер, пустые строки означают просто исчезновение снаряда при выстреле

_this call **shotGeometry** - использовать из ОС «fired» для получения подробной информации о выстреле. Возвращаемый массив:

- 0 - выпущенный снаряд(объект);
- 1 - его стартовая позиция;
- 2 - 3D вектор скорости (м/с);
- 4 - абсолютная скорость (м/с);
- 5 - горизонтальный угол полета (азимут, в градусах);
- 6 - вертикальный угол полета (угол места, в градусах)

Практика: «Примеры (часть 3).Intro»

[Назад к списку тем](#)

Поиск

[Array, ConditionList] call **bestFit**

альтернативный синтаксис:

[Array, ConditionList, Weights] call **bestFit** - выбрать элемент из Array, выполняющий максимальное число условий из списка ConditionList или выполняющий наиболее «весомый» набор условий, если заданы их весовые коэффициенты

ClassName call **csvQuery** – создает функцию для получения информации из основанной на использовании *CSV базы данных с частичной поддержкой ООП.

Разработана для получения доступа к информации из конфига.

У DASH_Library есть 4 «класса»: "ammo", "mag", "weapon" и "letter".

Скриптописатели могут создавать свои «классы» в stringtable.csv аддона/миссии/мода:

```
"Language", "English"
"Tag|base", "{_propName1}, defaultValue1, ... {_propNameN}, defaultValueN"
"Tag|Class1", "_propName1=5; _prop_Name_21={green}"
"Tag|Class2", "_base={Class1}; _propertyName19=[2,-4,0.5]"
```

Имя класса чувствительно к регистру, имена свойств - нет.
 Все имена свойств должны быть допустимыми и не повторяющимися именами локальных переменных. Если класс не существует – вернет пустую строку.
 Типы значений по-умолчанию: строка, число, сторона или смешанный массив из этих типов.
 Все вариации «класса» начинаются с определенного тега и вертикальной черты.
 Пример из стрингтейбла DASH_Library:

```
Language, English
...
"mag|base", "{_ammo},[_count],1,{_size},256,{_pic},{},{_norm},{},{_hd},{}"
...
"mag|JAM_AT4Rocket", "_ammo={JAM_AT4ammo};_size=2*256;_pic={\JAM_Magazines\pics\JAM_AT4Rocket.paa}"
...
"mag|JAM_CAVS_AT4Rocket", "_base={JAM_AT4Rocket};_ammo={JAM_CAVS_AT4ammo}]"
```

Одно свойство не должно использоваться в "конструкторе класса": `_base`.
 Оно обеспечивает наследование значений свойств (а набор свойств одинаковый).
 У класса `"mag|JAM_CAVS_AT4Rocket"` те же свойства, что и у его "отца",
`"mag|JAM_AT4Rocket"`, но параметр `"_ammo"` переопределен.
 Заметим, что значение `"_count"` в обоих случаях не упоминается – используется значение по-умолчанию из "конструктора" (=1).

3 следующие функции реализуют технологию [csvQuery](#) применительно к ПЕХОТНОЙ амуниции.
 Все они чувствительны к регистру аргумента

ClassName call [cfgAmmo](#) – возвращает массив свойств класса пуль/гранат **ClassName**

0 - `_sim` – параметр конфига *simulation*. По-умолчанию: «shotBullet»;
 1 - `_hit` – массив из параметров [hit, indirectHit, indirectHitRange].
 По-умолчанию: [0, 0, 0];
 2 - `_range` – массив [minRange, maxRange]. По-умолчанию: [0, 0];
 3 - `_ir` – IRLock. По-умолчанию: 0 (бот может атаковать пехоту);
 4 - `_air` – AirLock. По-умолчанию: 0 (бот не атакует самолеты и вертушки);
 5 - `_col` – не совсем как в конфиге – цвет для дыма или свечения, например:
 "red",
 "green",
 "yellow",
 "black",
 "violet",
 "white",
 "purple",
 "blue",
 "orange".
 По-умолчанию: {}

ClassName call [cfgMag](#) – возвращает массив свойств магазина **ClassName**

0 - `_ammo` – класс(ы) пуль/гранат – строка или массив. По-умолчанию: [];
 1 - `_count` – число пуль/гранат (размер обоймы). По-умолчанию: 1;
 2 - `_size` – размер магазина (число как в конфиге). По-умолчанию: 256;
 3 - `_pic` – путь к картинке. По-умолчанию: {};
 4 - `_norm` – для магазинов с повышенным разбросом (HD) -обычная версия.
 По-умолчанию: {};
 5 - `_hd` – для обычных магазинов – версия с повышенным разбросом (HD).
 По-умолчанию: {};

ClassName call [cfgWeapon](#) – возвращает массив свойств оружия **ClassName**

0 - `_name` – название (массив символов, не всегда как в конфиге).
 По-умолчанию: [];

1 - `_type` - тип оружия. Доступные значения:

- "AR" - автоматическая винтовка;
- "AR+GL" - автоматическая винтовка с подствольным гранатометом;
- "SMG" - автоматическое оружие под пистолетный патрон (пистолет-пулемет);
- "SMG+GL" - ПП с подствольным гранатометом;
- "GL" - легкий гранатомет вроде M79 или 6G30 (`primaryWeapon`);
- "AT" - противотанковое оружие вроде `RPGLauncher` или `CarlGustavLauncher`;
- "AA" - ПЗРК;
- "LMG" - легкий пулемет - занимает 1 слот;
- "MG" - пулемет, занимает слоты 1 и 16;
- "SHG" - дробовик;
- "R" - винтовка (неавтоматическое `primaryWeapon` без оптики);
- "SR" - снайперская винтовка, занимающая 1 слот;
- "AMR" - тяжелая снайперская винтовка (слоты 1+16);
- "P" - пистолет или револьвер;
- "MP" - автоматическое оружие из пистолетного слота (32) ;
- "LD" - лазерный дальномер / ЛЦУ;
- "RADIO" - рация;
- "RUCK" - рюкзак;
- "PROXY" - компонент статического оружия, вроде треноги от пулемета;
- "MELEE" - ножи, мечи и т.п.;
- "PUT" - закладывание зарядов;
- "THR" - бросание гранат и других предметов;

По-умолчанию: `{}`;

2 - `_side` - сторона, использующая это оружие. По-умолчанию: `side objectNull`;

3 - `_mz` - название второго ствола. По-умолчанию: `{}`;

4 - `_score` - `true`, если ПП/автомат/пулемет оснащен оптикой/коллиматором.

По-умолчанию: `false`;

5 - `_mag` - магазины для первого ствола (строка или массив). По-умолчанию: `[]`;

6 - `_mag2` - магазины для второго ствола вроде ГП25 или M203 (строка/массив).

По-умолчанию: `[]`;

7 - `_pic` - путь к картинке. По-умолчанию: `{}`;

8 - `_sd` - `true`, если оружие бесшумное. По-умолчанию: `false`

Заметка: `_mag` и `_mag2` из `cfgWeapon` могут быть обработаны `cfgMag`, а `_ammo` из результата работы `cfgMag` может быть обработан с помощью `cfgAmmo` - поэтому все эти параметры чувствительны к регистру

Еще одна технология, основанная на свойствах `stringtable.csv`: доступ к элементам глобального массива с помощью разделенных точками строковых ключей: функции `db*`.

Трехуровневая структура такого массива должна быть определена в стрингтейбле.

Переменные `_nodes` позволяют создать весь массив из значений (`call _init`), для элементов без этой переменной используется `[]`.

Имена ключей чувствительны к регистру, имена свойств-переменных - нет.

Хороший пример - собственный массив глобальных ресурсов `DASH_Library`:

```
"gdb|dash", "_get={dash};_set={};_nodes=[{addonflags},{propnames}]..."
"gdb|dash.propnames", "_get={dash select 4};_set={dash set[4,_this]}"
"gdb|dash.fog", "_get={dash select 18};_set={dash set[18,_this]};_nodes=[{now},{forecast},{changespeed}]"
"gdb|dash.fog.now", "_get={{dash select 18}select 0};_set={{dash select 18} set[0,_this]};_init={0}"
"gdb|dash.fog.forecast", "_get={{dash select 18}select 1};_set={{dash select 18} set[1,_this]};_init={0}"
```

`[stringKey, notArray] call dbAdd`

alternative syntax:

`[stringKey, [notArray1, notArray2, notArray3...]] call dbAdd`

- добавляет 1 или несколько значений к элементу глобальному массива `stringKey`

`[stringKey, notArray] call dbAddNew`

alternative syntax:

`[stringKey, [notArray1, notArray2, notArray3...]] call dbAddNew`

– добавляет 1 или несколько значений к элементу глобальному массива **stringKey**, причем только значения, которые не было раньше

stringKey call **dbCreate** – создает глобальный массив из значений по-умолчанию.

[stringKey, notArray] call **dbDelete**

alternative syntax:

[stringKey, [notArray1, notArray2, notArray3...]] call **dbDelete** – удаляет 1 или несколько значений из элемента массива

stringKey call **dbGet** – возвращает элемент глобального массива. Массивы возвращает по ссылке (можно изменять с помощью команд **set** и **resize**)

stringKey call **dbHint** – показывает содержимое элемента массива

[stringKey, anyValue] call **dbSet** – устанавливает элементу массива новое значение

[Array, Var] call **find2** – стандартная функция поиска, чувствительная к регистру. В CWA работает как команда **find**, в OFP медленнее, но также работает со смешанными массивами

[StringArray, String] call **findString** – возвращает номер строки **String** в массиве или -1 в случае неуспеха. Нечувствительна к регистру строк. В отличие от команды **in**, найдет "aK47" в (magazines guer1), если он там есть

[Array, Var] call **findAll** – вернет все индексы вхождения переменной **Var** в массив или [] в случае неуспеха. Чувствительна к регистру строк

[Matrix, ColNumber, Var] call **findInColumn** – вернет индекс вхождения переменной **Var** в столбце **ColNumber** вложенного массива **Matrix**. В случае неуспеха возвращает -1. Нечувствительна к регистру строк

[Array, Var] call **findInNested** – вернет индекс вхождения переменной **Var** в массив массивов или [] в случае неуспеха. Чувствительна к регистру строк

[NumericArray, Number] call **findInSorted** – возвращает индекс числа **Number** в массиве чисел **NumericArray**, отсортированном по-возрастанию. Работает быстрее других функций **find***, но массив должен быть подготовлен функцией **qSort** или **bSort**

NumArray call **max** – наибольшее число в массиве

NumArray call **min** – наименьшее число в массиве

[Array, Database] call **selectByKeys** – вернет записи (подмассивы) базы данных (вложенный массив) **Database** с ключами (0-ми элементами), соответствующими элементам **Array**. Функция чувствительна к регистру

[ExprWHERE, arrayFROM] call **sqlDelete** – удаляет записи из **arrayFROM**, соответствующие SQL-подобному запросу **exprWHERE**.

ExprWHERE содержит названия полей через запятую, также доступно имя **_i** – номер записи и возвращает true или false. Упомянутые в **exprWHERE** имена должны присутствовать в таблице **arrayFROM** и начинаться с префикса.

Пример **arrayFROM** с двумя записями:

```
[
  ["_Name", "_Armor", "_GunCaliber"],
  ["T55G", 300, 105],
  ["ZSU", 250, 23]
]
```

Пример **ExprWHERE**: { (_Name=="ZSU") or (_Armor<300) }.

Также можно использовать {true}, чтобы удалить все записи.

Функция не возвращает результата, но меняет исходный **arrayFROM**

[exprSELECT, arrayFROM] call sqlSelect

альтернативный синтаксис:

[exprSELECT, arrayFROM, exprWHERE] call sqlSelect

альтернативный синтаксис:

[exprSELECT, arrayFROM, exprWHERE, exprORDERBY] call sqlSelect - возвращает записи из **arrayFROM**, соответствующие SQL-подобному запросу.

ExprSELECT содержит названия полей через запятую, также доступно имя **_i** – номер записи. Пример **ExprSELECT**: **"_Name, _Armor, _MainCaliber"**. Также можно использовать **"**"**, чтобы получить полные записи.

Упомянутые в **exprSELECT** и в последующих выражениях имена должны присутствовать в таблице **arrayFROM** и начинаться с префикса.

Пример **arrayFROM** смотри [выше](#).

ExprWHERE – выражение, которое должно возвращать true или false, например

{(_Name=="ZSU") or (_Armor<300)}.

ExprORDERBY – выражение, которое должно возвращать число, используется для сортировки результата. По-умолчанию сортировка идет по-возрастанию, выражение типа **{-_Armor}** отсортирует записи по убыванию поля **_Armor**.

Исходная таблица **arrayFROM** не изменяется

Практика: «Примеры (часть 3).Intro»

[Назад к списку тем](#)

Самодельные обработчики событий

Или Custom Event Handlers (СЕН)

[Owner, EHType, CodeToRun] call AddCustomEH

альтернативный синтаксис:

[ArrayOfOwners, EHType, CodeToRun] call AddCustomEH

альтернативный синтаксис:

[Owner, EHType, CodeToRun, Path2ScriptFolder] call AddCustomEH

альтернативный синтаксис:

[ArrayOfOwners, EHType, CodeToRun, Path2ScriptFolder] call AddCustomEH

– добавляет юнита, группу или массив юнитов в цикл внутреннего скрипта, который выполняет выражение **CodeToRun** при каждом наступлении события типа **EHType** с его владельцем/владельцами. Функция сильно напоминает команду **AddEventHandler**.

Собственно, внутренний скрипт начинает выполняться с первого вызова **AddCustomEH** с данным типом ОС и прекращается с удалением последнего обладателя этого события. Добавление и удаление самодельных ОС происходит через глобальную очередь запросов.

В **CodeToRun** при срабатывании обычно передаются 3 значения:

_this select 0: владелец сработавшего ОС;

_this select 1: старое значение контролируемого параметра;

_this select 2: новое значение контролируемого параметра.

ОС **"weapons"** и **"magazines"** (срабатывают на изменения в экипировке юнита)

возвращают больше значений:

первые три – стандартные, а дальше

_this select 3: добавленное оружие/магазины;

_this select 4: брошенное оружие/магазины

Во избежание «тормозов» при массовом добавлении юнитам самодельных ОС

вместо **forEach** лучше использовать запуск с 0-м аргументом – массивом **ArrayOfOwners**.

Если приделать юниту 2 и более самодельных ОС одного типа, будет срабатывать только последний из них.

Доступные типы СЕН:

"ammoprimary" – изменение числа патронов в стволе **primaryWeapon**;

"behaviour" – изменение поведения;
 "rating" – изменения рейтинга;
 "canfire" – изменение способности стрелять;
 "canmove" – изменение способности двигаться;
 "canstand" – изменение способности стоять;
 "crew" – изменения в составе экипажа транспорта;
 "crew3" – изменения в составе действительного (водитель, стрелок и командир) экипажа транспорта;
 "freefall" – падение с высоты;
 "health" – изменение уровня здоровья;
 "magazines" – изменение набора магазинов в инвентаре;
 "primary" – смена основного оружия;
 "secondary" – смена дополнительного оружия;
 "weapons" – любая смена оружия, в том числе бинокля и пистолета.

Параметр **Path2ScriptFolder** позволяет скриптерам использовать свою папку с собственными ОС, изготовленными на базе имеющихся примеров.

Для упрощения редактирования, во всех СЕН-скриптах размечены 2 области для редактирования, остальное лучше не менять.

```

;Start modify 1 -----
_type="formation!"
_get={formation _this};
_delay=0.367
;End modify -----
  
```

_type – название обработчика, под которым он будет добавляться, удаляться или менять интенсивность запросов.

_get – выражение, вычисляющее через промежутки времени, примерно равные _delay, контролируемое значение, тип значения может быть любым.

Простейший случай, если результат – число, строка, сторона, группа или объект:

```

;Start modify 2 -----
?(_w!=_w2):[_ui, _w, _w2] call (_e select _i)
;End modify -----
  
```

_ui – с кем произошло (как-правило, юнит или группа), _w – старое значение, _w2 – новое значение. Для сравнения, редактируемые участки СЕН "weapons":

```

;Start modify 1 -----
_type="weapons"
_get={weapons _this};
_delay=0.896
;End modify -----

.....

;Start modify 2 -----
_w3=_w2-_w
_w4=_w-_w2
?(count(_w3+_w4)!=0):[_ui, _w, _w2, _w3, _w4] call (_e select _i)
;End modify -----
  
```

[Owner, EHType] call **RemoveCustomEH** – очень похожа на команду

unit RemoveAllEventHandlers type.

Owner (юнит или группа) будет «отчислен» из скрипта, обслуживающего тип события **ENType**. При массовом одновременном использовании может сильно тормозить игру

[**ENType**, **Number**] call **SetCEHDelay** – установить время, выделяемое на опрос юнитов обработчиком событий типа **ENType**. У каждого ОС есть рекомендуемое время запроса, которое можно сократить для более быстрой реакции на событие или увеличить в целях борьбы с торможением игры

Практика: «Примеры (часть 3).Intro»

[Назад к списку тем](#)

Системные

[**AnyVariable**, **StrComment**] call **debug** – сохраняет отладочную информацию для последующего показа с помощью **hintDebug**

Array_Of_Var_Names call **deleteGlobals** – удаляет указанные переменные. Явного результата не возвращает

[**SLX_Path**, **ECP_Path**] call **findLostVar** – возвращает индекс первой не инициированной переменной, или -1 в случае если все они определены. Поможет в отладке скриптов, использующих глобальные переменные

call **fw113** – true, если запущена Fwatch версии 1.13 или выше. Некоторые функции не работают без Fwatch, или реализуют не все свои возможности

call **getLanguage** – возвращает язык, используемый при данном запуске OFP. Возможные варианты: "Spanish", "Czech", "English", "Italian", "German", "French", "Russian". Неопознанные языки считает за "English"

call **hintDebug** – демонстрирует отладочную информацию, собранную через **debug**. Между сообщениями – задержка по времени

addon_name_lover_case call **isAddonLoaded** – проверяет, доступно ли дополнение или мод. Аргумент – строка в нижнем регистре. Моды всегда пишутся с «собакой»: "@bwod" call **isAddonLoaded**. База данных содержит более 220 сигнатур для аддонов и 20 для модов. Проверенные дополнения запоминаются, повторная проверка обычно мгновенная. Для файлов с собственной stringtable.csv или своим классом cfgAmmo выполняется очень быстро, для других может тормозить. Выводит сообщение, если аддон/мод не найден в базе

call **isCWA** – true, если запущен CWA, false, если ОФП

"VarName" call **isNil** – возвращает true, если переменная **VarName** не задана, либо уничтожена выражением **VarName = nil**. Имя переменной должно быть в кавычках или фигурных скобках!

Key call **optionGet** – возвращает значение из глобального ассоциативного массива

[**Key**, **NewValue**] call **optionSet** – устанавливает ключу глобального ассоциативного массива новое значение. Если такого ключа раньше не было – создаст

Var call **print** – форматирует **Var** в строку и печатает ее командой **sideChat**

[**PropOwner**, **StrPropName**] call **propGet** – возвращает значение определенного пользователем свойства **StrPropName** у его обладателя **PropOwner**. Обладателем может быть юнит, группа или даже заданный строкой фиктивный объект.

В качестве результата могут быть получены любые данные, но если не найден обладатель или у него нет такого свойства, то на выходе будет строка «undefined»

[PropOwner, StrPropName, Value] call **propsRemove** – удаляет указанные свойства у его обладателя PropOwner (объект, группа, строка или сторона)

[PropOwner, StrPropName, Value] call **propSet** – устанавливает свойство StrPropName равным Value его обладателю PropOwner (объекту, группе, строке или стороне)

[OwnersArray, p1, v1, p2, v2...] call **propSetM** – устанавливает каждому в массиве обладателей (обычно юнитов, но допускаются группы и строки) набор определенных пользователем свойств и их значений. Успешно тестировалось присвоение 361 юниту по 50 свойств разного типа, в том числе со значениями – объектами, строками и массивами

Expression call **test** – печатает вместе выражение **Expression** и его (отделенный двойной сплошной линией) явный результат с помощью команды hintC. Массивы изображаются с отступами для демонстрации их структуры, максимальное число строк при показе массива – 32, остальное заменяется троеточием.

Если явного результата нет, например, сделано присвоение, или один из аргументов в **Expression** был равен nil – печатает только **Expression**.

«Противопоказания» - смотри **varTypeSafe**.

При массовом применении в скриптах нужно ставить ~2 или хотя бы ~0.01 между вызовами **test**

Variable call **varType** – возвращает тип переменной, если она является числом ("number"), строкой ("string"), массивом ("array"), логическим значением ("bool"), стороной ("side"), объектом ("object"), группой ("group") или триггером ("trigger").

Не использовать со строками длиной > 2000 символов(2Kb) – будет падение CWA/OFP. К большим массивам относится спокойно

Variable call **varTypeSafe** – возвращает тип переменной, если она является числом ("number"), строкой ("string"), массивом ("array"), логическим значением ("bool"), стороной ("side"), объектом ("object") или группой ("group").

Не вызывает крушение CWA на длинных строках, но «спотыкается» на камерах и (если не включена Fwatch) еще и на триггерах – выдает сообщение об ошибке

call **version** – возвращает номер версии DASH_Library

[Condition, Xpression] call **whileDo** – работает аналогично циклу **while Condition do Xpression** но, в отличие от него, не имеет ограничения в 10 000 итераций. Для аналогичных целей иногда лучше подходит FOREACH

Практика: «Примеры (часть 1).Intro», «Тестовая кампания»

[Назад к списку тем](#)

Случайные числа

NumArray call **rndCase** – случайное целое в диапазоне [0, (count NumArray)-1].

Чем больше элемент в **NumArray** – тем больше вероятность получить его номер **_i** на выходе. Вероятность появления каждого **_i** равна **NumArray[_i] / SUM(NumArray)**

Array call **rndElement** – случайный равновероятный выбор элемента из массива

[Array, NumArray] call **rndElementPro** – случайный выбор элемента массива с неравными шансами. Вероятность выбора **_i**-го элемента из **Array**:

NumArray[_i] / SUM(NumArray)

[Min, Max] call **rndFloat** – случайное вещественное число между **Min** и **Max**.

Max call **rndInt** – случайное целое число между 0 и Max-1

call **rndLandPos** – случайная позиция на суше. Работает для островов с ненулевым числом статичных объектов на карте

[Array, N] call **rndSample** – N случайных элементов из Array, выбранные индексы не повторяются

[Array, N] call **rndSample2** – N случайных элементов из Array, выбранные индексы могут повторяться

Array call **shuffle** – случайное перемешивание массива. Меняет исходный массив.

NestedArray call **shufflePivoted** – **NestedArray** перемешивается, сохраняя соответствие позиций своих подмассивов, т.е. во всех подмассивах будет проведена одинаковая случайная перестановка.
Нет явного результата – изменяет исходный массив

Практика: «Примеры (часть 2).Intro»

[Назад к списку тем](#)

Сортировка

Все перечисленные ниже функции (за исключением **merge2sorted**) не возвращают явного результата, но изменяют порядок элементов в исходных данных

NumArray call **bSort** – сортировать массив по-возрастанию. Алгоритм – пузырьковая сортировка. Очень медленный на больших массивах, но оптимальный на массивах длиной менее 10 элементов

NumArray call **bSort2** – сортировать 2 массива по-возрастанию элементов в первом. Алгоритм – пузырьковая сортировка. Очень медленный на больших массивах, но оптимальный на 2-х массивах длиной менее 10 элементов каждый

NumArray call **qSort** – сортировать массив по-возрастанию.
Алгоритм – быстрая сортировка

[Array, Sample] call **qSortByExample** – сортировать элементы **Array** в том порядке, в котором они входят в образец **Sample**, например в массив букв алфавита.
Алгоритм – быстрая сортировка

[Array, Expression] call **qSortByExpr** – сортировать **Array** в порядке возрастания выражения **Expression**, применяемого к каждому его элементу.
Алгоритм – быстрая сортировка

[NumArray, Array1, Array2...ArrayN] call **qsortM** – сортировать массивы по возрастанию элементов в первом из них. Алгоритм – быстрая сортировка

[Array, I, J] call **idxSwap** – поменять местами 2 элемента в массиве

Array call **inverse** – перевернуть исходный массив. Можно применять после сортирующих функций, если нужен порядок по-убыванию

[Array1, Array2] call **merge2sorted** – объединяет 2 отсортированных по-возрастанию массива, создает новый отсортированный массив

Практика: «Примеры (часть 3).Intro»

[Назад к списку тем](#)

Среда обитания

Building call **countBuildingPos** - количество доступных позиций в здании **Building**

call **countStatic** - число статичных объектов на карте

call **daylight** - возвращает 0, если вызвана ночью, 1 - если днем и 0.5 - если в сумерках. Точность - порядка 10 секунд. Для выбора соответствующего значения, например из массива строк, делаем так:

```
["ночь", "сумерки", "день"] select (2*call daylight)
```

call **fog** - текущий уровень тумана.

DASH_library при штатном запуске устанавливает уровень тумана 0, под Fwatch 1.13+ туман устанавливается в соответствии с прогнозом погоды в mission.sqm. Если в дальнейшем вместо команды **setFog** использовать функцию **setFog2**, то функция **fog** будет работать корректно

[Pos, R] call **groundSlope**

альтернативный синтаксис:

[Pos, R, Step] call **groundSlope** - средний уклон и направление максимального уклона в радиусе R метров от позиции **Pos**, опционально можно задать шаг поиска в градусах

[Position, Radius] call **inForest** - true, если не далее чем в **Radius** метрах от указанной позиции находится лес (отдельные деревья не в счет!).

Полезно проверить, нет ли поблизости леса, когда надвигаются несколько танковых взводов...

Obj call **inTown** - возвращает true, если объект в городе/деревне.

Для мелких деревень без доступных для входа строений может соврать

Pos call **inWater** - возвращает true, если позиция покрыта водой

call **isWinter** - возвращает true, если запущена зимой.

Точность определения сезона - порядка недели

call **mapName** - возвращает расширение папки текущей миссии как команда **getWorld**, но работает также для 200 островов в OFP 1.96

Pos call **nearestVillage** - возвращает массив длины 3:

0 - название ближайшего населенного пункта;

1 - позиция его центра;

2 - расстояние по карте от **Pos** до его центра.

Работает на тех островах, которые есть в базе данных

call **objectID** - возвращает номер статичного объекта на карте, вернет 0 для всего, что установлено через редактор миссий (определено в mission.sqm).

Требует Fwatch 1.13+!

call **overcast** - уровень облачности.

DASH_library при штатном запуске устанавливает облачность 0.5, под Fwatch 1.14 облачность устанавливается в соответствии с прогнозом погоды.

Если в дальнейшем вместо команды **setOvercast** использовать функцию **setOvercast2**, то **overcast** будет возвращать верные значения

[Pos, Ra] call **overObstacle**

альтернативный синтаксис:

[Pos, Ra, ObjectArray] call **overObstacle** - возвращает true если в радиусе Ra от позиции **Pos** есть препятствие, препятствующее посадке вертолета на этой площадке. Можно задать список игнорируемых типов препятствий

call **rain** - интенсивность дождя. DASH_library при своем запуске устанавливает

интенсивность дождя 0, если в дальнейшем вместо команды `setRain` использовать функцию `setRain2`, то функция `rain` будет работать корректно

Number call `setTime` - установить время на **Number** часов

[Sec, Level] call `setFog2` – аналогична команде `Sec setFog Level`, но позволяет в дальнейшем определять уровень тумана функцией `fog`.
Время изменение уровня тумана задается в секундах, уровень тумана может быть в диапазоне от 0 (нет вообще) до 1 (всё в тумане)

[Sec, Level] call `setOvercast2` – аналогична команде `Sec setOvercast Level`, но позволяет узнавать текущий уровень облачности функцией `overcast`.
Время изменение уровня облачности задается в секундах, уровень облачности может быть в диапазоне от 0 (нет вообще) до 1 (сплошные тучи)

[Sec, Level] call `setRain2` – аналогична команде `Sec setRain Level`, но позволяет узнать текущий уровень дождя, он определяется функцией `rain`.
Время изменение силы дождя задается в секундах, сила дождя может быть в диапазоне от 0 (нет дождя) до 1 (ливень)

CharVectorVillage call `villagePos` – возвращает 2-D позицию населенного пункта **CharVectorVillage** несмотря на возможные опечатки. Применение ограничено островами из базы данных (более сотни). На острове без городов возвращает []

CharVectorVillage call `villageInfo` – возвращает массив из 3-х элементов [правильное имя, ошибка (0...1), 2-D позиция] для данного населенного пункта несмотря на возможные опечатки. Применение ограничено островами из базы данных (более сотни). На острове без городов возвращает []

call `wind` – выдает массив из 2-х значений:
0 - текущее направление ветра в градусах;
1 - скорость ветра в метрах в секунду

call `windVector` – выдает массив из 2-х значений:
0 - скорость ветра по оси X (м/с);
1 - скорость ветра по оси Y (м/с)

Практика: «Примеры (часть 2).Intro»

[Назад к списку тем](#)

Текст

Country call `alphabet` – получить символьный массив, содержащий алфавит данной страны. Возможны 6 вариантов: "gb", "us" "ru" - массив в нижнем регистре, "GB", "US" "RU" – в верхнем, для любого другого ключа – пустой массив

[Array, strDelimiter] call `array2Report`

альтернативный синтаксис:

[Array, strDelimiter, bUseEnds] call `array2Report` – форматировать массив в виде отчета, используя строку-разделитель `strDelimiter` и опционально – окончания для множественного числа

[Array, FormatSingle, FormatPlural, Delimiter] call `array2ReportFMT` – форматировать массив в виде отчета, используя строку `FormatSingle` для форматирования единичных элементов, `FormatPlural` – для множественных и строку `Delimiter` в качестве разделителя. В строку `FormatSingle` через «%1» передается элемент массива, в строку `FormatPlural` через «%1» передается элемент, а через «%2» -его количество в массиве

Array call `array2string`

альтернативный синтаксис:

[Array, StrDelimiter] call **array2string** - преобразовать массив в строку, элементы будут разделены пробелом или строкой **StrDelimiter**

[Array_of_CharArrays, CharArray] call **catAfter**

альтернативный синтаксис:

[Array_of_Strings, String] call **catAfter** - каждый элемент массива строк или массивов дополнить справа строкой или массивом

[Array_of_CharArrays, CharArray] call **catBefore**

альтернативный синтаксис:

[Array_of_Strings, String] call **catBefore** - каждый элемент массива строк или массивов дополнить слева строкой или массивом

CharArray call **cv2string** - из символьного массива получает строку.

Может использоваться для слияния любого массива строк, но «заточен» под короткие

[CV1, CV2] call **editDistance** - расстояние редактирования - мера различия двух символьных векторов произвольной длины, показывает «стоимость» исправления одного слова на другое. Ошибки типа лишней или пропущенной буквы «стоят» 2.

Выражение для стоимости замены символа задается глобально с помощью **fzTextProps**, по-умолчанию - {1}, в общем виде - функция 2-х аргументов

[CV1, CV2] call **endsWith**

альтернативный синтаксис:

[CV1, CV2, bIgnoreCase] call **endsWith** - возвращает true, если символьный вектор **CV1** заканчивается символьным вектором **CV2**. Опционально задается параметр «игнорировать регистр символов» (true/false), по-умолчанию регистр игнорируется

Unit call **firstName** - имя юнита (без фамилии). Требуется Fwatch 1.13+

[CV1, CV2] call **cosineCVCVDistance** - косинусное расстояние между словами, возвращает число от 0 до 1, показывающее косинус-меру (**cos_dist**) между результатами применения **symbolCounts** к каждому из аргументов. Работает с массивами символов произвольной длины. Преимущество: нечувствительна к перестановкам букв в слове. Как следствие, недостаток: анаграммы вроде ["e", "a", "t"] и ["t", "e", "a"] всегда дают результат 0 (полное совпадение)

[String, CV] call **FuzzyStrCVCompare** - сравнивает строку **String** и массив символов **CV**, возвращает true если между ними не более 1 несовпадения. При наличии Fwatch рекомендуется использовать **fzTextMatch**

[Text1, Text2] call **fzTextMatch**

альтернативный синтаксис:

[Text1, Text2, MaxError] call **fzTextMatch** - возвращает true, если нечеткое сравнение двух текстов дает ошибку не более **MaxError**. Доступные методы сравнения: "edit"- **editDistance** и "cosine"-**cosineCVCVDistance**. Тексты могут быть строками или символьными векторами.

С запущенной Fwatch аргументы-строки разбиваются на символы и после сравниваются по глобально заданному методу.

Без нее 2 символьных массива сравниваются аналогично ("edit" или "cosine"), 2 строки сравниваются через ==, а в смешанном случае [CV, String] или [String, CV]- применяется **FuzzyStrCVCompare** - так себе метод, но лучше, чем ничего

[Array1, Array2] call **inStr** - возвращает позицию в массиве **Array1**, начиная с которой в нем содержится массив **Array2**, -1 в случае несовпадения. Разработана для символьных массивов, но также работает с числовыми

Unit call **lastName** - фамилия юнита. Требуется Fwatch 1.13+

CharArray call **lCase** - преобразовать массив символов в нижний регистр

CharArray call **lCaseRu** - версия **lCase** для кириллицы

CharArray call **noDoubleLetters** – удалить двойные (тройные и т.д.) символы из символьного массива. Возвращает новый массив

Number call **nth** – натуральное число – в порядковое числительное:
1 ==> "1st", 12 ==> "12th"

CV call **removeOFPEctag**: делает разбитое на символы название юнита/оружия/магазина более подходящим для рапортов за счет отбрасывания тега. Теги с подчеркиванием распознаются всегда, теги без него, например **["K", "E", "G"]** проверяются по базе

[StrArray1, StrArray2] call **removeStrings** – удаляет один массив строк из другого, игнорируя регистр.

[CV, CV_Old, CV_New] call **replaceCV** – заменить 1-ю найденную последовательность символов **CV_Old** в массиве символов **CV** на последовательность **CV_New**.
Возвращает измененный массив

[Array, Delimiter] call **split** – разбивает массив **Array** на подмассивы, используя аргумент **Delimiter** как разделитель

[CV1, CV2] call **startsFrom**

альтернативный синтаксис:

[CV1, CV2, bIgnoreCase] call **startsFrom** – возвращает true, если символьный вектор **CV1** начинается с вектора **CV2**. Опционально задается параметр «игнорировать регистр символов» (true/false), по-умолчанию регистр игнорируется

String call **string2CV** – разбиение строки на отдельные символы

[CV, StartIdx, EndIdx] call **subStr** – возвращает новый массив, содержащий символы исходного массива от позиции **StartIdx** до позиции **EndIdx**.
Работает и с массивами любых других типов

Char Vector call **symbolCounts** – подсчитывает, сколько раз в массиве символов встретились каждая из 26 букв и 10 цифр. Возвращает массив целых чисел длины 36

CharArray call **uCase** – преобразовать массив символов в верхний регистр

CharArray call **uCaseRu** – версия **uCase** для кириллицы

Практика: «Примеры (часть 2).Intro»

[Назад к списку тем](#)

Транспорт

Veh call **canFire2** – модификация команды **canFire**, проверяет также наличие оружия и боекомплекта. Бинокль, ПНВ и гудок за оружие не считаются

Veh call **canMove2** – модификация команды **canMove**, проверяет также наличие топлива в баке

Veh call **cargo** – массив юнитов, находящихся в транспортном отсеке техники **Veh**

Veh call **effectiveCommander** – юнит, непосредственно командующий в технике **Veh**

UnitArray call **enableQueryVehicles** – создает внутренний массив юнитов для работы функции **getVehicles**. Не нужна при использовании Fwatch 1.13+

VehType call **getVehicles**

альтернативный синтаксис:

ArrayOfTypes call **getVehicles** – лучше всего работает в Fwatch 1.13, возвращает

массив техники (транспорт и объекты), представленный потомками данного класса машин/объектов или машинами точно совпадающими с одним из классов. Для работы без Fwatch требуется большой триггер типа Any с условием Present и функция **enableQueryVehicles**. С Fwatch работает в редактируемых миссиях и распакованных кампаниях

Veh call hasCommanderPlace – имеет ли данный транспорт место командира?

Veh call hasDriverPlace – имеет ли данный транспорт место водителя?

Veh call hasGunnerPlace – имеет ли данный транспорт место стрелка?

[Unit, VehInfo] call knownVehicles

альтернативный синтаксис:

[Unit, VehInfo, Knowledge] call knownVehicles

альтернативный синтаксис:

[Unit, VehInfo, Knowledge, BlackList] call knownVehicles – найти все машины или объекты известные Unit, принадлежащие заданному классу VehInfo или дочерние от него, либо точно совпадающие с одним из классов в VehInfo, имеющие величину Unit knowsAbout _x не менее, чем Knowledge (если не задана – 1.4) и не принадлежащие «черному списку» объектов, если такой список задан. Результат – массив объектов и массив их «известности», который можно использовать для сортировки по-убыванию значения knowsAbout. Для работы без Fwatch требуется большой триггер типа Any с условием Present и функция **enableQueryVehicles**. С Fwatch работает в редактируемых миссиях и распакованных кампаниях

[PosInfo, VehInfo] call nearbyVehicles

альтернативный синтаксис:

[PosInfo, VehInfo, Radius] call nearbyVehicles

альтернативный синтаксис:

[PosInfo, VehInfo, Radius, BlackList] call nearbyVehicles – найти все машины или объекты около PosInfo (юнит или позиция), принадлежащие заданному классу VehInfo или дочерние от него, либо точно совпадающие с одним из классов в VehInfo, удаленные от PosInfo не более, чем на Radius (если не задан - 50 метров) и не принадлежащие «черному списку» BlackList (по-умолчанию - пустой). Результат – массив объектов и массив расстояний до них, который можно использовать для сортировки по удаленности. Для работы без Fwatch понадобится большой триггер типа Any с условием Present и функция **enableQueryVehicles**. С Fwatch работает в редактируемых миссиях и распакованных кампаниях

[PosInfo, VehInfo] call nearestVehicle

альтернативный синтаксис:

[PosInfo, VehInfo, Radius] call nearestVehicle

альтернативный синтаксис:

[PosInfo, VehInfo, Radius, BlackList] call nearestVehicle – найти ближайшую машину или объект около PosInfo (юнит или позиция), принадлежащую заданному классу VehInfo или дочернюю от него, либо точно совпадающую с одним из классов в VehInfo, удаленную от PosInfo не более, чем на Radius (если не задан - 50 метров) и не принадлежащую «черному списку» BlackList, если такой список задан. Результат – объект. Для работы без Fwatch нужен большой триггер типа Any с условием Present и функция **enableQueryVehicles**. С Fwatch работает в редактируемых миссиях и распакованных кампаниях

[Veh, NewSpeed] call setSpeed – мгновенно установить скорость транспорта Veh равной NewSpeed

Unit call vehicleRole - позиция юнита в занимаемом им транспорте.

Возможные результаты: "Driver", "Gunner", "Commander", "Cargo", "None"

UnitArray call vehicles - возвращает массив техники, занятой представителями массива юнитов

Практика: «Примеры (часть 3).Intro»

[Назад к списку тем](#)

БЛАГОДАРНОСТИ

Некоторые из представленных функций – это переименованные и/или переработанные варианты кода опытных скриптописателей.
У отдельных джедаев были санионерены позаимствованы их идеи.
Респект всем перечисленным специалистам по OFP и командам таких спецов:

Baddo – `countBuildingPos`;

команда BINMOD - `dirOfMove`, `vehicleRole`;

Bn880 – `varType`;

банда The Chain of Command - `transpose` и идея нейронной сети для OFP;

DenVdmj – `formatOut`, `findLostVar`, `varType`, `varTypeSafe`, `invQueryCargo`, константа `sideUnknown` и пример тестовой кампании всего из одной миссии;

Dschulle – `grid2pos`;

отряд мода ECP – концепция глобального массива ресурсов;

Faguss - `qsort`, `qsortM`, `fw113`, `string2CV`, техническая поддержка по Fwatch;

FOX2 – `pos2grid`;

Fragor1 – идея определяемых пользователем свойств объектов;

General Baron - `findString`, `squadNumber`, `squadOrder`;

Igor Drukov – `relPos2d`;

разработчики мода Invasion44 – концепция «баз данных» на основе функций;

KTottE - `pop`, `pushNew`;

группировка LIBMOD – `cargo`;

Mandoble – `inFOV`;

персонал проекта MCAR – идея использовать `stringtable.csv` для хранения функций;

Mr.Peanut - `rndFloat`, `elevation`, `highestLowest`;

Raptorsaurus - `elevationAngle`, `inForest`, `groundSlope`;

создатели мода SLX - `radioState`;

snYpir – `dir2obj`;

Spooner – `getLaserDot`;

Toadlife – `watchDir`;

Vectorboson - `groupIsCargo`, `getGroupSpeed`, `getGroupSpread`,
`mostInjured`, `filterGroups`, `nth`, `shuffle`, `array2report`
+идея модуля `Rnd.sqf`, теперь добавленного в стрингтейбл;
+помощь в доработке `isNil`, `rndFloat` и `varType`;

Отдельное спасибо:

DenVdmj – перевод справочника по скриптовым командам от BIS + подробный справочник по скриптам;

Dschulle – программа PBOX;

Faguss – доработка программы Fwatch и техническая поддержка;

Kegetys – создание программы Fwatch;

Lone~Wolf – плагин к Notepad++ для подсветки команд и названий объектов OFP;

команда Notepad++ – создали лучший в мире блокнот;

Razorwings18 – программа OFPDialogueMaker;

Rinzzza – фундаментальный справочник по OFP

ОБРАТНАЯ СВЯЗЬ

Вопросы, сообщения об ошибках и предложения по теме функций и скриптов для ОФП/СВА присылаем на ящик andrey.melekhoff@mail.ru

Аккаунт на **Flashpoint.ru** и **OFPEC.com**: SoldierEPilot